## REMARKS

Claims 1, 4, 7, 10, 11, 17, 18, 19, 22, 23, 24, 26, 31 and 32 are amended. Claims 1-34 remain in the application for consideration. In view of the following remarks, Applicant respectfully requests reconsideration and allowance of the subject application.

### Specification Amendment

Applicant has amended the specification to insert the appropriate application serial numbers of the list of related applications.

### 35 U.S.C. § 101 Rejections

Claims 17-28 stand rejected under 35 U.S.C. § 101 as being directed to non-statutory subject matter. Independent claims 17 and 22 have been amended, as suggested by the Office, to insert "computer-implemented" before the term "method" as suggested by the Office.

### 35 U.S.C. § 112 Rejections

Claims 1-34 stand rejected under 35 U.S.C. § 112, second paragraph as being indefinite. More specifically, the Office argues that the claims are indefinite as the parameters M, N, T and V do not recite numerical ranges. Applicant has amended relevant occurrences of these parameters to recite that the parameters are "greater than 0" as noted by the Office. Applicant thanks the Office for the Office's helpful suggestion. In view of the amendments, the Office's rejection is traversed.

In addition, the Office argues that claims 29-30 are indefinite as it is unclear whether these claims are directed to storage mediums, computer systems or method claims because they depend on a method claim. Applicant respectfully disagrees. For example, consider claim 29 reproduced just below:

> 29. A ***storage medium*** comprising a plurality of executable instructions which, when executed, implement a method according to claim 22.

This claim is very clearly and unmistakably directed to a storage medium that comprises instructions that implement the method of claim 22. Thus, this claim is not a ***method*** claim. This claim is known as a dependent Beauregard claim which is a widely used and accepted form of claim.

In addition, the Patent Office has approved of this type of claim as evidenced by the large number of patents that have been granted which include dependent Beauregard claims. As an example, the Office is respectfully directed to the following patents and their associated claims:

- U.S. Patent No. 6,738,666 – claim 12
- U.S. Patent No. 6,738,512 – claim 34
- U.S. Patent No. 6,725,262 – claim 33

Claim 30 is directed to a ***computing system*** and is written in a form that is similar to that of claim 29. For all of the reasons that claim 29 is not indefinite, this claim is not indefinite.

**35 U.S.C. §§ 102 and 103 Rejections**

Claims 1-10, 12-18, 20-25 and 27-34 stand rejected under 35 U.S.C. § 102(b) as being anticipated by U.S. Patent No. 5,913,038 to Griffiths.

Claims 11, 19 and 26 stand rejected under 35 U.S.C. § 103(a) as being obvious over Griffiths in view of U.S. Patent No. 5,790,935 to Payton.

Before discussing the substance of the Office's rejection, a short discussion of Griffiths and of Applicant's disclosure is provided to assist the Office in appreciating the patentable distinctions between the cited references and the claimed subject matter.

**The Griffiths Reference**

Griffiths is directed to the construction of a graph by automatically connecting filters to perform processing operations upon data streams representing a variety of multimedia data formats. Griffiths' disclosure describes assembling the graph by selecting appropriate filters that can handle the data processing requirements for the desired data stream(s). As Griffiths instructs, a graph can be constructed by (1) selecting a set of filters, including an appropriate file reader compatible with the media type of the data stream(s), a demultiplexer or parser for separating multiplexed data, a decoder for decoding encoded data, and a renderer to display or sound the data, and (2) combining these filters within the architecture of a filter graph to efficiently process the multimedia data.

Griffiths' approach can be appreciated by the text that appears in column 25 starting at around line 50. Specifically, Griffiths describes how a filter graph is assembled through a number of enumerated steps, which are reproduced just below:

1. Parse the source file to select a source filter that can accept as an input a one of the data streams of the source file, and load the source filter.
2. Select an output of source filter.
3. For the selected output of the source filter, select and connect a new filter that can accept the selected output from the source filter as an input to the new filter.
4. If the connection is successfully completed, then proceed to step 5; otherwise disconnect the new filter and repeat step 3 for another filter.
5. In the event that the newly connected filter has an output, then select the output of this connected filter.
6. For the selected output of the newly connected filter, select and connect another filter that can accept the selected output as an input to the new filter.
7. If the connection is successfully completed, then proceed to step 8; otherwise disconnect the new filter and repeat step 6 for another filter.
8. Repeat steps 5-7 until the selected data stream is rendered.
9. Repeat steps 2-8 for each remaining output of the source filter to render any remaining data streams of the source file.

What Griffiths describes throughout its disclosure (and indeed in the enumerated steps above) is the process of ***initially*** building a filter graph for rendering a project. Applicant's disclosure builds upon the filter graph concept and extends the technology of filter graphs in a manner that permits dynamic graph building during execution of the filter graph.

**Applicant's Disclosure**

Applicant's dynamic graph building approach is probably best appreciated from its Figs. 40-44 and the related discussion in the Specification starting on page 51 under the heading "Dynamic Graph Building".

More specifically, Applicant's disclosure describes an approach in which a render engine dynamically loads filter graph chains as they are needed. Aspects of the approach can also discard, or cache processing chains when they are no longer required to support execution of the processing project.

To illustrate the benefits afforded by dynamic graph building, assume, for example, that an editing project included over 100 sources, yet only three (3) of them were ever required at any given time to support execution of the filter graph. As noted in the Specification, loading three sources will be executed much fast than 100 sources, thereby permitting execution of the filter graph to commence much more rapidly than conventional filter graph implementations. Further, the memory and processing resources required to support three (3) sources will generally be less than those required to support 100 sources.

**Fig. 40** is a flow chart of an example method for processing media content, in accordance one embodiment. More particularly, Fig. 40 illustrates an example method in which a render engine dynamically generates and manages a filter graph to reduce the computational and/or memory requirements placed on a host system. As shown, method 4000 begins with block 4002, wherein the render engine receives an indication to generate a development project. According to one implementation, the render engine receives the indication from a higher-level application, e.g., media processing application, to assist a user in generating a processing project (e.g., a media processing project).

In block 4004, the render engine identifies the number and nature of the media sources within the user-defined processing project, in preparation for generating a filter graph representation of the processing project. As introduced earlier in the Specification, for each of the identified sources, the render engine determines the necessary transform filters required to pre-process the source (i.e., the source processing chain), preparing the chain for presentation, in one embodiment, to a matrix switch filter (described earlier in the Specification).

As the Specification instructs, unlike conventional implementations which would proceed to generate the entire filter graph in preparation for execution of the processing project, the render engine generates a ***list of sources*** and ***when they are required*** in the filter graph. An example of a data structure comprising a list of processing chains is presented with reference to Fig. 41.

Turning briefly to **Fig. 41**, a graphical illustration of an example data structure comprising a processing chain execution list is presented. As shown, the chain execution list 4100 is comprised of a number of information fields, e.g., 4102-4110 which detail, in part, which chains are required at a particular time in project execution. In accordance with the illustrated example embodiment of Fig. 41, chain execution list 4100 is depicted comprising a chain identifier field 4102, a source identifier field 4104, a project time field 4106, a source time field 4108, and a dependencies field 4110.

Upon identifying a project source and the associated filters required for pre-processing the source (i.e., the source chain), the render engine assigns the chain an identifier which uniquely identifies the source chain within the context of the filter graph. Accordingly, the chain execution list 4100 includes a field 4102 which maintains a list of the chains utilized in the associated project. Within the

context of the filter graph, the chain identifier corresponds to the source and the associated pre-processing filters.

The source identifier field 4104 contains information denoting the project source associated with a particular chain identifier. In this regard, the source identifier field 4104 may well contain a file name, a file handle, or any other suitable source identifier.

The project time field 4106 denotes at what point during project execution the source chain is required. The source time field 4108 denotes what portion of the source file is required to support execution of the processing project. As instructed by the Specification, it should be appreciated that a user may well utilize the whole source file or any part thereof, as defined by the processing project.

The dependencies field 4110 denotes whether the associated chain is dependent upon any other chain. That is, multiple chains may rely on a common source and/or a subset of another source chain. In certain implementations, it would not be advantageous to unload source chains prior to their execution and/or the execution of chains dependent thereon. Accordingly, the render engine maintains a list of such dependencies within the chain execution list 4100. As noted in the Specification, it is to be appreciated, however, that certain circumstances may arise where it is necessary to unload a chain prior to or during execution, or prior to execution of an otherwise dependent chain. One such example is where the processing project utilizes a hierarchical structure, wherein individual chains are assigned a priority level. An implementation is contemplated, for example, wherein the priority of a particular chain is dynamically managed by a matrix switch filter within a filter graph based, at least

in part, on how soon the chain is required to support the uninterrupted execution of the processing project, i.e., chains which are required more urgently are assigned a higher priority and, as a result, are processed at the disadvantage of other, lower priority chains. In the extreme, lower priority chains are unloaded to enable loading of a higher priority chain. As instructed by the Specification, it is to be appreciated that, although depicted as a two-dimensional data structure, chain execution lists of greater or lesser complexity may well be substituted.

Returning to Fig 40 and, in particular, block 4006, render engine 222 dynamically generates and manages a filter graph representation of the processing project ***invoking only those chains associated with sources that are necessary to support the current and/or impending execution of the filter graph***. It is to be appreciated that by not opening each of the chains of a processing project, the render engine reduces the amount of memory required to build the filter graph, thereby reducing the amount of memory required to complete execution of the project, i.e., recall the example where the entire graph utilized 100 sources, but only required three (3) at any given time. An example method of dynamically generating and managing a filter graph is presented with reference to the flow chart illustrated in Fig. 42.

Turning to **Fig. 42**, an example method for dynamically generating and managing a filter graph is presented, in accordance with one embodiment. In accordance with the illustrated example implementation of Fig. 42, method 4006 commences with block 4202 in which the render engine determines which chains are required to fulfill execution of the development project for the next M seconds. According to this example implementation, M must be greater than the minimum time it takes to completely load the next chain. In accordance with the illustrated

example implementation, wherein a matrix switch filter controls the pace of project execution, the matrix switch filter provides an indication to the render engine of what chains are required in block 4202.

According to one implementation, M is dynamically generated based on a number of factors including, but not limited to, processing speed, available memory, the complexity of the development project, the number and type of the source chains, and the like. In certain implementations, the processing system maintains a performance history (not shown), and dynamically modifies the processing threshold M based on past performance. According to one implementation, M is stochastically set to ten (10) seconds. Accordingly, the render engine maintains only the chains currently required to support the next ten seconds of execution. It is important to note that project execution does not necessarily correlate to rendering of the composite generated by the filter graph. That is, in certain implementations, execution of the filter graph is performed as fast as possible, utilizing the shared memory resources of the matrix switch filter 308 to buffer the composite until the rendering chain consumes the composite.

In block 4204, the render engine determines whether a threshold of loaded chains (e.g., a maximum chain-count) (T) has been exceeded. In certain implementations, the number of loaded chains will be limited due to memory limitations. According to one example implementation, setting T equal to one (1) is popular in that it requires the render engine to analyze the filter graph for chains that are no longer required (e.g., exhausted chains) whenever a new chain is considered for loading. According to one example implementation, the maximum number of loaded chains supported by the render engine is seventy (70). Accordingly, once the render engine has identified the chains required (block

4202), a determination is made of whether there is space in which to load them into the filter graph (block 4204).

If the chain count threshold (T) has not yet been reached, the render engine loads the identified chains, block 4206. The matrix switch filter will initiate execution of the newly loaded chains to fulfill the execution requirements of the development project.

If, in block 4204 the chain-count threshold (T) has been reached, the render engine determines whether one or more chains may be unloaded from the filter graph. Thus, in block 4206, the render engine identifies any currently loaded chains that will not be utilized in the next N seconds. Source chains may be accessed multiple times to process multiple portions of an associated source. Thus, in accordance with steps 4202-4206, a source chain may have been loaded to meet an impending execution requirement, and remains loaded to satisfy a subsequent processing task. However, where resources are running short, the render engine along with matrix switch filter(s) determine which chains are not required in the next N seconds and, in block 4210, instructs the render engine to unload the identified chains. As above, N may well be dynamically derived based on past performance. In accordance with one example implementation, N is thirty (30) seconds. According to one implementation, the render engine determines whether the chains will be required for subsequent processing in the current or a future filter graph. If so, the filter chain is removed from the active filter graph by the render engine and cached for subsequent re-integration in this or a future filter graph.

In block 4212, the render engine determines whether unloading of the identified chain(s) in block 4210 has brought the total chain-count below the

threshold a cutoff threshold (V). According to one implementation, V is greater than T. This is particularly useful if T has been set to one (1), as described above. If so, processing continues with block 4206 as the render engine loads the chains identified in block 4202.

If, in block 4212, the chain-count threshold is still exceeded, the render engine re-analyzes the current filter graph and identifies the lowest priority chains, block 4214. That is, the filter graph may well be comprised of seventy chains, all of which will be required in the next thirty (N) seconds. If, however, the chains identified in block 4202 are needed prior to any of the seventy chains currently loaded in the filter graph, those chains are assigned a lower priority. Processing continues with block 4210 as the lower priority chains are unloaded, as render engine 222 re-analyzes the chain-count, in block 4212. If the filter graph has space available, processing continues with block 4206, else it continues with block 4214.

**Fig. 43** graphically illustrates an example data structure utilized to manage dynamic graph building, according to one example implementation. In accordance with the illustrated example embodiment of Fig. 43, a filter graph 4300 is presented. ***Unlike conventional filter graph implementations, wherein all chains 4302-4308 would be loaded prior to execution of the development project, filter graph 4300 illustrates the dynamic nature of the described embodiments.*** In the illustrated example of Fig. 43, matrix switch filter 308 has identified at least two source chains 4302, 4304 which are required in the next M seconds to support the timely processing of the development project. Such chains are illustrated in Fig. 43 with a solid black line to denote that these chains are currently loaded into the filter graph 4300. In accordance with one aspect, the development project may

well contain additional chains (e.g., 4306 and 4308) that will be required to complete execution of the development project, but which are not yet required and are, thus, not yet loaded. Such chains are illustrated in Fig. 43 with dotted lines, denoting that they are not currently loaded into the filter graph 4300. By limiting the number of currently loaded chains to a threshold (T) or, alternatively (V), the present implementation reduces the memory requirements necessary to satisfy even the most complex of development projects by unloading chains when they are no longer required, without stifling the user's creativity by artificially limiting the size of the filter graph.

As should be apparent from the above description, Applicant's disclosure pertains to dynamic graph building. The various embodiments described and claimed in the present application are each associated with some aspect of dynamic graph building.

## The Claims

**Claim 1** has been amended and recites a system comprising [added language appears in bold italics]:

- a plurality of sources; and
- an interface, selectively coupled to the plurality of sources, to generate and implement a development project of processing chains ***at least one chain of which comprises multiple filters***, wherein the interface loads a processing chain for each of the plurality of media sources at a point during the execution of the project when the chain is required, and wherein the interface is configured to unload at least a subset of the chains when they are not required.

In making out the rejection of this claim, the Office argues that Griffiths anticipates this claim and cites to various sections in support thereof. Applicant respectfully disagrees. Nonetheless, Applicant has amended this claim to clarify that at least one of the processing chains comprises multiple filters. As such, the interface is configured, as recited in the claim, to unload a subset of the chains when they are not required, which subset can include a chain that comprises multiple filters. Perhaps at this point, reference to Fig. 43 and the related discussion in the Specification (reproduced above) will illustrate one example of what is intended to be covered by this claim.

In making out the rejection of this claim, the Office argues that Griffiths discloses unloading at least a subset of the chains when they are not required—citing to column 21, lines 1-10 for support. There, Griffiths describes simply removing a ***single filter*** in its graph building process and not a ***chain*** of filters as contemplated in this claim.

Accordingly, for at least this reason, Griffiths does not anticipate this claim and it is allowable.

**Claims 2-16** depend from claim 1 and are allowable as depending from an allowable base claim. In addition, as Griffiths does not anticipate claim 1, the Office's further reliance on Payton in making out the rejection of claim 11 is not seen to add anything of significance.

**Claim 17** has been amended and recites a computer-implemented method for generating and managing a development project comprising [added language appears in bold italics]:

- identifying processing chains required to support execution of the development project over the next M seconds; and
- loading the identified processing chains as long as a currently loaded chain-count does not exceed an initial threshold, T, ***wherein T and M are greater than 0.***

In making out the rejection of this claim, the Office argues that Griffiths anticipates this claim citing to column 23, line 34 through column 24, line 67 (for anticipating the recited act of "identifying"), and to column 23, lines 3-19 (for anticipating the recited act of "loading").

In making out the rejection of this claim, the Office argues that it interprets "next M seconds" as the amount of time required to execute each recursion level of the Griffiths reference. This interpretation ignores the specifically recited claim language. Specifically, the claim recites "identifying processing chains ***required to support execution of the development project*** over the next M seconds". Applicant can find no disclosure that mentions identifying processing chains required to support execution of the development project over the next M seconds. The amount of time required to execute each recursion level in Griffiths does not identify processing chains required to support execution of the development project over the next M seconds.

Further, the claim recites "loading the identified processing chains as long as a ***currently loaded chain-count*** does not exceed an initial threshold, T". The Office argues that Griffiths' processing (which, when the recursion count is less than a predetermined threshold, conducts a search for an available filter) anticipates this element. Apparently, the Office is equating Applicant's recited

chain count with Griffiths' recursion count. Applicant respectfully submits that the Griffiths' recursion count has nothing to do with the recited chain count.

Accordingly, this claim is not anticipated by Griffiths and is allowable.

**Claims 18-21** depend from claim 17 and are allowable as depending from an allowable base claim. In addition, as Griffiths does not anticipate claim 17, the Office's further reliance on Payton in making out the rejection of claim 19 is not seen to add anything of significance.

**Claim 22** has been amended and recites a computer-implemented method for managing a media processing project comprising [added language appears in bold italics]:

- identifying each of a plurality of sources required to satisfy the media processing project;
- determining when one or more chain(s) associated with each of the plurality of sources is required to support execution of the media processing project; and
- selectively loading and unloading each of the chains during execution of the filter graph based, at least in part, on when each of the chains are required to support execution of the media processing project, ***at least some selectively loaded and unloaded chains comprising multiple filters***.

In making out the rejection of this claim, the Office argues that Griffiths anticipates the claim. Applicant respectfully disagrees. Nonetheless, Applicant has amended the claim to clarify that at least some of the selectively loaded and unloaded chains comprise multiple filters. The excerpt of Griffiths cited to by the Office and argued to anticipate the act of "selectively loading and unloading" (i.e. column 21, lines 1-10) simply discusses unloading a single filter and not a chain comprising multiple filters. Accordingly, this claim is not anticipated by Griffiths.

**Claims 23-30** depend from claim 22 and are allowable as depending from an allowable base claim. In addition, as Griffiths does not anticipate claim 22, the Office's further reliance on Payton in making out the rejection of claim 26 is not seen to add anything of significance.

**Claim 31** recites a storage medium comprising a plurality of executable instructions which, when executed, implements an interface to manage development and execution of a development project, wherein the interface identifies processing chains required to support execution of the development project over the next M seconds, and loads the identified processing chains as long as a ***currently loaded chain-count*** does not exceed an initial threshold, T, wherein M and T are greater than 0.

In making out the rejection of this claim, the Office argues that this claim is rejected for reasons similar to those used to reject claim 17. Apparently, the Office again assumes that Griffiths' recursion count is the same as Applicant's recited "currently loaded chain-count". As noted above, this is simply not the case. Accordingly, for at least this reason, this claim is not anticipated by Griffiths and is allowable.

**Claims 32-34** depend from claim 31 and are allowable as depending from an allowable base claim.

## Conclusion

Applicant respectfully submits that all of the claims are in condition for allowance and Applicant respectfully requests a Notice of Allowability be issued forthwith. If the next anticipated action is to be anything other than issuance of a

Notice of Allowability, Applicant respectfully requests a telephone call for the purpose of scheduling an interview.

Respectfully Submitted,

Dated: 7/1/07

By: _____

Lance R. Sadler
Reg. No. 38,605
(509) 324-9256